Filter List Generation for Underserved Regions

Alexander Sjösten
Chalmers University of Technology

Peter Snyder
Brave Software

Antonio Pastor
Universidad Carlos III de Madrid

Panagiotis Papadopoulos
Brave Software

Benjamin Livshits
Brave Software
Imperial College London

ABSTRACT

Filter lists play a large and growing role in protecting and assisting web users. The vast majority of popular filter lists are crowd-sourced, where a large number of people manually label resources related to undesirable web resources (e.g. ads, trackers, paywall libraries), so that they can be blocked by browsers and extensions.

Because only a small percentage of web users participate in the generation of filter lists, a crowd-sourcing strategy works well for blocking either uncommon resources that appear on "popular" websites, or resources that appear on a large number of "unpopular" websites. A crowd-sourcing strategy will perform poorly for parts of the web with small "crowds", such as regions of the web serving languages with (relatively) few speakers.

This work addresses this problem through the combination of two novel techniques: (i) deep browser instrumentation that allows for the accurate generation of request chains, in a way that is robust in situations that confuse existing measurement techniques, and (ii) an ad classifier that uniquely combines perceptual and page-context features to remain accurate across multiple languages.

We apply our unique two-step filter list generation pipeline to three regions of the web that currently have poorly maintained filter lists: Sri Lanka, Hungary, and Albania. We generate new filter lists that complement existing filter lists. Our complementary lists block an additional 3,349 of ad and ad-related resources (1,771 unique) when applied to 6,475 pages targeting these three regions.

We hope that this work can be part of an increased effort at ensuring that the security, privacy, and performance benefits of web resource blocking can be shared with all users, and not only those in dominant linguistic or economic regions.

CCS CONCEPTS

• **Security and privacy** → *Human and societal aspects of security and privacy*; • **Information systems** → *Web applications.*

KEYWORDS

ad blocking, filter lists, crowdsource

ACM Reference Format:
Alexander Sjösten, Peter Snyder, Antonio Pastor, Panagiotis Papadopoulos, and Benjamin Livshits. 2020. Filter List Generation for Underserved Regions. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3366423.3380239

1 INTRODUCTION

Hundreds of millions of web users (i.e. 30% of all internet users [25]) use filter lists to maintain a secure, private, performant, and appealing web. Prior work has shown that filter lists, and the types of content blocking they enable, significantly reduce data use [37], protect users from malware [36], improve browser performance [28, 38] and significantly reduce how often and persistently users are tracked on the web.

Most filter lists are generated through crowd-sourcing, where a large number of people collaborate to identify undesirable online resources (e.g. ads, user tracking libraries, anti-adblocking scripts etc..) and generate sets of rules to identify those resources. Crowd-sourcing the generation of these lists has proven a useful strategy, as evidenced by the fact that the most popular lists are quite frequently used and frequently updated [33, 41].

The most popular filter lists (e.g. EasyList, EasyPrivacy) target "global" sites, which in practice means either websites in English, or resources popular enough to appear on English-speaking sites *in addition to* sites targeting speakers of other languages. Non-English speaking web users face different, generally less appealing options for content blocking. Web users who visit non-English websites that target relatively wealthy users generally have access to well maintained, language-specific lists. Indeed, the French [9], German [11], and Japanese [17] specific filter lists are representative examples of well-maintained, popular filter lists targeting non-English web users. Similarly, linguistic regions with very large numbers of speakers also generally have well maintained filter lists. Examples here include well maintained filter lists targeting Hindi [15], Russian [21], Chinese [5], and Portuguese [4, 20] websites.

Sadly, users who visit websites in languages with fewer speakers, or with less wealthy users, have worse options. Put differently, the usefulness of crowd-sourced filter lists depends on having a large or affluent crowd; filter lists targeting parts of the web with less, or less affluent, users are left with filter lists that are smaller, less well-maintained, or both. Visitors speaking these less-commonly-spoken languages have degraded web experiences, and are exposed to all the web maladies that filter lists are designed to fix.

Compounding the problem, in many cases, users in these regions are the ones who could benefit most from robust filter lists, as network connections may be slower, data may be more expensive, the frequency of undesirable web resources may be higher. An example which motivates this work and illustrates the inability of current filter lists to adequately block ads on a regional website in Albania can be seen in the screenshot in Figure 1. In this example,

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380239

we browse the website *gazetatema.net* while using AdBlock Plus (which uses EasyList, a "global" targeting filter list).

While there has been significant prior work on automating the generation of filter lists [24, 26, 29, 34], this existing work is focused on replicating and extending the most popular English and globally-focused filter lists, with little to no evaluation on, or applicability to, non-English web regions. In this paper, we target the problem of improving filter lists for web users in regions with small numbers of speakers (relative to prominent global languages). We select three regions as representative of the problem in general: Albania, Hungary and Sri Lanka, using a methodology presented in Section 4.1.

We describe a two-pronged strategy for identifying long-tail resources on websites that target under-served linguistic regions on the web: (i) a classifier that can identify advertisements in a way that generalizes well across languages, and (ii) a method for accurately determining how advertisements end up in pages (as determined by either existing filter lists or our classifier), and by using this information, generate new, generalized filter rules.

We use this novel instrumentation to both build *inclusions chains* (i.e. measurements of how every remote resource wound up in a web page), and determine how high in each inclusion chain blocking can begin. This allows us to (i) generate generalized filter rules (i.e. rules that target scripts that include ad images on each page, instead of rules that target URLs for individual advertisements), and to (ii) ensure we do not block new resources that will break the website in other ways.

Contributions. In summary, this paper makes the following contributions to the problem of blocking unwanted resources on websites targeting audiences with smaller linguistic audiences.

(1) The implementation and evaluation of an **image classifier for automatically detecting advertisements on the web** which relies on a mix of perceptual and contextual features. This classifier is designed to be robust across many languages (and particularly those overlooked by existing research) and achieves accuracy of 97.6 % in identifying images and iframes related to advertising.
(2) Novel, open source **browser instrumentation, implemented as modifications to the Blink and V8 run-times** that allows for determining the cause of every web request in a page, in a way that is far more accurate than existing tools. This instrumentation also allows us to accurately attribute every DOM modification to its cause, which in turn allows us to predict whether blocking a resource would break a page.
(3) The design of a **novel, two stage pipeline for identifying advertising resources on websites**, using the previously mentioned classifier and instrumentation, to identify long-tail advertising resources targeting web users who do not speak languages with large global communities.
(4) A **real world evaluation** of our pipeline on sites that are popular with languages that are (relatively) uncommon online. We find that our approach is successful in significantly improving the quality of filter lists for web users without large, language-specific crowd sourced lists. As our evaluation shows, our generated lists block an additional 3,349 of ad and ad-related resources (1,771 unique) when applied to 6,475 pages targeting these three regions.

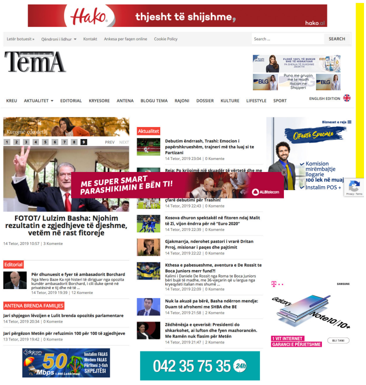
Hako thjesht të shijshme
TemA
ME SUPER SMART PARASHIKIMIN E BËN TI!
FOTOT/ Lulzim Basha: Njohim rezultatin e zgjedhjeve të djeshme, vetëm në rast fitoreje
Oferta Speciale
50 Mbps
042 35 75 35

Figure 1: Motivating example of current filter lists' regional inefficiency. Screenshot of Albanian website browsed with Adblock Plus.

2 SOLUTION REQUIREMENTS

A successful contribution to the problem of improving the quality of filter lists in small web regions should account for the following issues:

Scalability. The primary difficulty of generating effective blocking rules for small-region web users is the reduced number of people who can participate in a crowd-sourced list generation. While portions of the web are targeted at large audiences (e.g. sites in English language, or web regions with a large number of language speakers) can count on a large number of users to report unwanted resources, or generally distribute the task of list generation, regions of the web targeting only a small number of users (e.g. languages with less speakers) do not have this luxury. A successful solution therefore likely requires some kind of automation to augment the efforts of regional list generators.

Generalize-ability. In most of the cases, ads are rendered by scripts. In addition, every time an ad-slot is filled, the embedded ad image may have come from a different URL. Approaches that directly target the URLs serving ad-related resources are then likely to become stale very quickly. An effective solution to the problem would instead target the "root cause" of the unwanted resources being included in the page, in this case the script, which determines what image URLs to load. Approaches that attempt to only build lists of URLs of ad-related images are therefore unlikely to be useful solutions to the problem for the long term (as seen also from the screenshot in Figure 1).

Web compatibility. Content blocking necessarily requires modifying the execution of a page from what the site-author intended, to something hopefully more closely aligned with the visitor's goals and preferences. Modifying the page's execution in this way (e.g. by changing what resources to load, by preventing scripts from executing, etc.) frequently cause pages to break, and users to abandon content blocking tools. While filter lists targeting large audiences can rely on the crowd to report breaking sites to the list authors, (so that they can tailor the rules accordingly), filter lists targeting smaller parts of the web often do not have enough users to maintain this positive feedback loop. An effective system for programmatically augmenting small-region filter lists must therefore take extra care to ensure that new rules will not break sites.

3 METHODOLOGY

This section presents a methodology for programmatically identifying advertising and other unwanted web resources in under-served regions. This section proceeds by describing (i) a high-level overview of our approach, (ii) a hybrid classifier used to identify image-based web advertisements, (iii) unique browser instrumentation used in our approach, (iv) how we identify ad-libraries and other "upstream" resources for blocking, (v) how we determined if a request was safe to block (i.e. would not break desirable functionality on the page), and (vi) how we generated filter list rules from the gathered ad URLs.

3.1 Overview

Our solution to improving filter lists for under-served regions consists of the combination of two unique strategies. First, we designed a system for programmatically determining whether an image is an ad, in a cross-language, highly precise way. We use this classifier to identify ad images that are missed by crowd-sourced filter list maintainers.

Second we developed a technique for identifying additional resources that should be blocked, by considering the request chains that brought the ad into the page, and finding instances where we can block earlier in that request chain. We then apply this "blocking earlier in the chain" principle to both ads identified by existing filter lists, and new ads identified by our classifier, to maximize the number of resources that can be safely blocked. This approach also allows us to generate generalized blocking rules that target the *causes* of ads being included in the page, instead of only the "symptoms": the specific, frequently-changing image URLs.

We note that this approach could be applied to any region of the web, including both popular and under-served regions. However, since popular parts of the web are already well-served by crowd-sourced approaches, we expect the marginal improvement of applying this technique will be greatest for under-served regions, where there are comparatively few manual labelers.

The following subsections describe the implementation of each piece in our filter list generation pipeline. Section 4 describes the evaluation of how successful this approach was at generating new filter list rules for under-served regions.

3.2 Hybrid Perceptual/Contextual Classifier

First, our approach requires an oracle for determining if a page element is an advertisement, without human labeling. To solve this problem, we designed and trained a unique hybrid image classifier that considers both the image's pixel data, and page context an image request occurred in, when predicting if a page element is an advertisement. Our classifier targets both images (i.e. `<img>`) and sub-documents (i.e. `<iframe>`). Our classifier prefers precision over recall, since for filter list it is more important to *only* block ads, instead of blocking *every* ad.

3.2.1 Comparison to Existing Approaches. While there is significant existing work on image based (i.e. perceptual) web ad classification, we were not able to use existing approaches for two reasons. First, we had disappointing results when applying existing perceptual classifiers to the web at large. The existing approaches we considered did very well on the data sets they were trained on, but did a relatively poor job when applied to new, random crawls of the web.

Second, we were concerned that relying on perceptual features alone would reduce the classifier's ability to generalize across languages. We expected that adding contextual features (e.g. the surrounding elements in the page, whether the image request was triggered by JavaScript or the document parser, attributes on the element displaying the image) would make the classifier generalize better.

3.2.2 Classifier Design. Our approach combines both perceptual and contextual page features, each building on existing work. The perceptual features are similar to those described in the Percival [40] paper, while the contextual features are extensions of those used in the AdGraph [34] project. The probability estimated by the perceptual module is then used as an input to the contextual classifier.

Perceptual Sub-module. The perceptual part of our classifier expands Percival's SqueezeNet based CNN into a larger network, ResNet18 [30]. While the Percival project used a smaller network for fast online, in-browser classification, our classifier is designed for offline classification, and so faces no such constraint. We instead use the larger ResNet18 approach to increase predictive power. Otherwise, our approach is the same as that described in [40].

Contextual Sub-Module. The contextual part of our classifier does not consider the image's pixel data, but instead how the image loaded in the web page, and the context of the page the image or subdocument would be displayed in. Examples of contextual features include whether the resource being requested is served on the same domain as the requesting website, and the number of sibling DOM nodes of the `img` or `iframe` element initiating the request. These features are similar to those described in the AdGraph paper, and detailed in Figure 4. The browser instrumentation needed to extract these features is described in detailed in Section 3.3.

3.2.3 Classifier Evaluation. We built our image classifier in two steps. First we built a purely perceptual classifier, using approaches described in existing work. Second, when we found the perceptual classifier did not generalize well when applied to a new, independent sampling of images, we moved to a hybrid approach. In this hybrid approach, the output of the perceptual classifier is just one feature among many other contextual features. We found this hybrid approach performed much better on our new, manually labeled, random crawl of the Alexa 10k. The rest of this subsection describes each stage in this process.

Initially, we built a classifier using an approach nearly identical to the perceptual approach described in [40]. We evaluated this model on a combination of data provided by the paper's authors, augmented with a small amount of additional data labeled by ourselves. This data set is referred to in Figure 3 as the "Initial Alexa 10k Set". When we applied the training method described in [40] to this data set, we received very accurate results, reported in Figure 2.

Later, while building the pipeline described in this paper, we generated a second manually labeled data set of images and frames, randomly sampled a new crawl of the Alexa 10k. This data set is referred to in Figure 3 as "Alexa 10k Recrawl", and was collected

	Initial Alexa 10k Data Set			Alexa 10k Recrawl		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Perceptual-only	95.9 %	95.5 %	96.4 %	77.0 %	48.8 %	87.4 %
Hybrid	-	-	-	97.6 %	92 %	75 %

Figure 2: Comparison of classification strategies. "Perceptual-only" refers to the approach by Percival [40] and variants (best numbers reported). "Hybrid" uses both perceptual and contextual features, and performed much better on our independent sampling of images and frames from the Alexa 10k, especially with regards to precision.

	# Images	% Ads	# Frames	% Ads
Initial Alexa 10k Set	7,685	48 %	465	77 %
Alexa 10k Recrawl	2,610	14 %	1,034	41 %

Figure 3: Comparison of the distribution of ads for images and frames collected in each data set.

Content features
Height & Width
Is image size a standard ad size?
Resource URL length
Is resource from subdomain?
Is resource from third party?
Presence of a semi-colon in query string?
Resource type (image or iframe)
Perceptual classifier ad probability
Structural features
Resource load time from start
Degree of the resource node (in, out, in+out)
Is the resource modified by script?
Parent node degree (in, out, in+out)
Is parent node modified by script?
Average degree connectivity

Figure 4: Partial feature set of the contextual classifier.

between 2-6 months later than the previous data set[1]. When we applied the prior purely-perceptual approach to this new data set, we received greatly reduced accuracy. Most alarming of which, for our purposes, was the dramatically reduced precision. These numbers are also reported in Figure 2.

We concluded that perceptual features alone were insufficient to handle the breath of advertisements found on the web, and so wanted to augment the prior perceptual approach with additional, contextual features we expected to generalize better, both across languages and across time. A subset of these contextual features are presented in Figure 4, and are heavily based on the contextual ad-identification features discussed in the AdGraph [34] project.

After constructing our hybrid classifier from the combination of perceptual and contextual features, we achieved greatly increased precision, though at the expense of some recall. We used a Random Forest approach to combine the perceptual and contextual features, and after conducting a 5-fold cross-validation, achieved mean precision of 92 % and mean recall of 75 %, again summarized in Figure 2. Our hybrid classifier could not be evaluated against the initial Alexa 10k data set because the data set 1) programmatically determined some labels, and 2) was collected without our browser instrumentation, meaning we could not extract the required features.

3.3 Browser Instrumentation

In this subsection we present PageGraph, a system for representing and recording web page execution as a graph. PageGraph allows us to correctly attribute every page modification and network request to its cause in the page (usually, the responsible JavaScript unit). We use this instrumentation both to extract the contextual features described in Section 3.2.2, and to accurately understand what page modifications and downstream requests each JavaScript unit is responsible for.

Our approach is similar to the AdGraph [34] project, but is more robust (i.e. corrects categories of attribution errors) and broader (i.e. cover an even greater set of page behaviors). PageGraph is implemented as a large set of patches and modifications to Blink and V8 (approximately 12K LOC). The code for PageGraph is open source and actively maintained, and can be found at [19], along with information on how other researchers can use the tool.

The remainder of this subsection provides a high-level summary of the graph-based approach used by PageGraph, and how it differs from existing work.

3.3.1 Graph Representation of Page Execution. We use PageGraph to represent the execution of each page as a directed graph. This graph is available both at run-time, and offline (serialized as graphml [22]) for after-the-fact analysis. PageGraph uses nodes to represent elements in a web page (e.g. DOM elements, resources requested, executing JavaScript units, child frames) and edges representing the interaction between these elements in the page (e.g. an edge from a script to a node might depict the script modifying an attribute on the node, an edge from a DOM element node to a resource node might depict a file being fetched because of a `img` element's `src` attribute, etc.). All such page behaviors in the top-level frame, and child-local-frames, are captured in the graph.

We use PageGraph's context-rich recording of page execution for several purposes in this work. First, it allows to accurately and efficiently understand how a JavaScript unit's execution modified the page; we can easily determine which scripts made a lot of modifications to the page, and which had only "invisible" effects to e.g. fingerprint the user. Second, the graph allows us to determine

[1] the date range here is due to the majority of this data set being collected by the Percival authors, 6 months before our work, with a smaller additional amount of data being collected by ourselves later on.

how each element ended up in a page. For example, the graph representation makes it easy to determine if an image was injected in the page by a script, if so *what* other script, and how that script was included in the page, etc. Being able to accurately determine what page element is responsible for the inclusion of each script, frame or image element is particularly valuable to this work, as described in the following subsections.

3.3.2 *Differences from Existing Work.* The most relevant related work to PageGraph is the AdGraph project, which also modifies the Blink and V8 systems in Chromium to build a graph-representation of page execution. PageGraph differs from AdGraph in several significant ways.

Improved Attribution Accuracy. PageGraph significantly improves cause-attribution in the graph, or correctly determining which JavaScript unit is responsible for each modification. We observed a non-trivial number of corner cases where AdGraph would attribute modifications to the wrong script unit, such as when the script was executed as a result of an element attribute (e.g. `onerror="do_something()"`), or when the JavaScript stack is reset through events like timer callbacks (e.g. `setTimeout(do_something,1)`). PageGraph correctly handles these and a large number of similar corner cases.

Increased Attribution Breadth. PageGraph significantly increases the set of page events tracked in the graph, beyond what AdGraph records. For example, PageGraph tracks image requests initiated because of CSS rules and prefetch instructions, records modifications made in local sub-documents, and tracks failed network requests, among many others. This additional attribution allows for greater understanding of the context scripts execute in.

3.4 Generalizing Filter Rules

We next discuss how we generate generalized filter rules from the data gathered by the previously described image classifier and browser instrumentation. The general approach is to find URLs serving ad images and frames using the classifier, use the browser instrumentation to build the entire request chain that caused the advertisement to be included in the page (e.g. the script that fetched the script that inserted the image), and then again use the browser instrumentation to determine how far up each request chain we can block without breaking the page.

We build these request chains for both images (and frames) our classifier identifies as an ad, and for resources identified by network rules in existing filter lists (i.e. EasyList, EasyPrivacy and the most up to date applicable regional list). The former allows us to generalize the benefits of our image classifier, the latter allows us to maximize the benefits of existing filter lists.

3.4.1 *Motivation.* Blocking higher in the request chain has several benefits. First, and most importantly, targeting URLs higher in the request chain yields a more consistent set of URLs. While the specific images that an ad library loads will change frequently, the URL of the ad library itself will rarely change. Approaches that target the frequently changing image URLs will result in filter list rules that quickly go stale; rules that target ad library scripts (as one example) are more likely to be useful over time, and to a wider range of users. Moving higher in the request chain means we are more likely to programmatically identify ad libraries *in addition to* frequently changing, one-off image URLs.

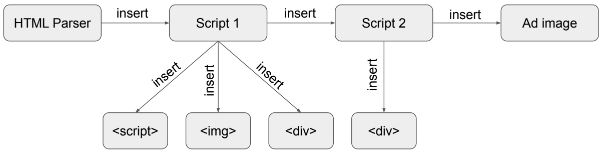

Figure 5: Example of a request chain, ending in an inserted ad image.

Second, blocking higher in request chains reduces the total number of requests, bringing privacy and performance improvements. Blocking a single "upstream" ad library may prevent the browser from needing to consider several "downstream" requests.

3.4.2 *Building Request Chains.* To generate optimized filter list rules, we target not only the ad images and ad frames in each page, but the scripts that injected those images and frames (and, potentially, the scripts that injected those scripts, etc.). We refer to the cause of a request as being "upstream", and the thing being requested "downstream". We refer to the list of elements that participated in an advertisement being included as its "request chain."

For each `<img>`, `<iframe>` and `<script>` in a page, we determine the request chain as follows:

(1) Locate each element in the PageGraph generated graph structure. Call this element `X`.
(2) Use the graph edges to determine how `X` was inserted in the document. If `X` was inserted by the parser (i.e. it appeared in the initial HTML text) then stop.
(3) Otherwise, append the script element `X` into the request chain for `X`, set the responsible script element as the new `X` and continue from #2 above.

A simplified result of this process is depicted in Figure 5. The figure shows a simplified request chain, where a script was included in the initial HTML ("script one"), that script programmatically inserted another script element into the document ("script two") and that second script inserted an advertising image into the page.

We use these request chains to determine the optimal place to start blocking, using the approach described in the following section.

3.5 Safe Blocking in Request Chains

This subsection describes how we determine whether blocking a script request is likely to break a page. We use this technique to determine how "high" in each request chain we can block, with the goal of determining the earliest "upstream" request we can block in a request chain without breaking the page. Our approach is "conservative" (i.e. prefers false negatives over false positives), under the intuition that users would prefer a working, ad-filled page, over a broken, ad-less page.

3.5.1 *Determining Page Breakage.* We use a pair of simple heuristics to determine whether blocking a script is likely to break a page. These heuristics are designed to distinguish scripts that only inject ads into pages from scripts that perform more complex, and hopefully user serving, page operations.

(1) If a script creates more than two subtrees in the document, we consider it **unsafe** to block.

(2) If a script inserts another script that matches condition #1, we consider it **unsafe** to block.
(3) We consider all other scripts **safe** to block.

Less formally, if a script makes no modifications to the structure of a page, or the modifications to the page are isolated to one or two parts of the page (e.g. one or two ads, an ad and an "ad choices" annotation, etc.) we consider it safe to block. Scripts that add elements to more than two parts of the page, or include scripts that do the same, are considered too risky to block, and too likely to break desirable page functionality.

We note that this is only a heuristic, one that matches our experience building and debugging advertising and tracking-blocking tools, but still only a heuristic. Although heuristics can be fooled by an attacker, they are being used in current tools to identify unwanted code. If the script for injecting ad content is updated to evade the heuristic, the heuristic can be updated. We choose the conservative figure of allowing modifications to a maximum of two regions of the page to favor false negatives over false positives (i.e. we'd rather allow an ad than break a page). The larger problem of predicting whether any given page modification breaks a site *in the subjective determination of the browser user* is an open research question, and one that would be its own complicated project.

3.5.2 Application to Filter List Generation. We use the above-described heuristics to determine the highest point in a request chain that can be blocked. For each request chain describing how an advertising image or frame was included in a page, we select the "highest" script request we can block, that will not break the page. Put differently, we want to select the earliest point in each chain to block, that will have no "downstream" breaking scripts. If there are no elements in the request chain that can be blocked, the last loaded resource (i.e. the ad) will be blocked only.

As a demonstration, consider Figure 5. Our system would generate two filter rules for this request chain, one targeting the "ad image", and one targeting "script 2". Our system begins by considering the most "downstream" request, the image element at the far right. This image has been identified as an ad, either by our classifier, or by existing filter lists. Using the browser instrumentation described in Section 3.3, we build the request chain for this image.

Next, we try to consider the earliest point in the request chain we can begin blocking. We observe that "script 2" only modifies one other part of the document (inserting a single `<div>` element) and so we consider this script safe to block. The next element in the request chain, "script 1" inserts elements into more than two parts of the document, and so we consider it "unsafe" for blocking.

3.6 Rule Generation

Finally, we describe how we turn the set of identified ad-serving URLs into filter list rules. We do so through the following four steps for each URL we determine to be blockable:

(1) Reduce the URL's domain to its `eTLD+1` root.
(2) Remove the query parameter portion of the URL.
(3) Remove the fragment portion of the URL.
(4) Remove the protocol from the URL.

We then record the modified (i.e. reduced) version of each URL as a right-rooted AdBlock Plus format filter rule [2]. For example, the URL `https://a.good.example.com/ad.html?id=3` would be recorded as `||example.com/ad.html`, and would block requests to `https://good.example.com/another-ad.html` and `http://a.b.good.example.com/ad.html?id=4`, but would not block requests to `https://other.domain.com/ad.html`. This approach is designed to generalize some (i.e. match other similar requests, even when irrelevant details like tracking related query parameters change) but not so much so that unrelated materials served on the same host are blocked.

Country	# Rules	# Network Rules	Last Update	Source
Albania	201	118	8	[1]
Hungary	1,407	662	3	[16]
Sri Lanka	69	42	22	[3]

Figure 6: Regional crawling data. The "Last Update" column gives the number of months since the last update, relative to October 2019.

Country	Commits	Start (Month-Year)	Average	Source
Albania	3	02-2019	0.27	[1]
Hungary	542	12-2014	8.9	[16]
Sri Lanka	16	03-2016	0.35	[3]
India	1,637	05-2018	81.85	[13]
Germany	11,982	01-2014	166.4	[12]
Japan	1,687	05-2014	24.8	[14]

Figure 7: Filter list activity. "Average" gives the average number of commits per month since the start of the git repo, relative to Jan. 2020.

4 EVALUATION

In this section, we evaluate the approach to regional filter list generation described in Section 3 by applying the technique to three representative under-served web regions. We find that the technique is successful, and generates 1310 new rules that identify 1,771 advertising URLs missed by existing filter lists. These new rules, when applied in addition to existing filter list rules, results in 27.1% more advertising resource being blocked than when using existing filter lists alone. We also find that our technique is useful in all three measured regions, though to varying degrees.

This section proceeds by first describing how we selected the three under-served regions used in this evaluation, then presents the crawling methodology we used to measure popular websites in each selected region, and follows by presenting the results of applying our methodology to each of these regions. The section concludes by presenting the output of our measurements (i.e. the newly generated filter list rules), so that they can be used by existing content blockers.

4.1 Selecting Regions for Evaluation

We evaluated our approach on three regions under-served by existing crowd-sourced filter lists: Albania, Hungary, and Sri Lanka. We selected these regions after looking for regions that matched four criteria.

(1) The national language was not a major world language.
(2) The amount of updates to the regional supplementary list is significantly lower compared to more popular lists.

Country	# Domains	# Pages	# Images	# Frames
Albania	740	1,805	38,763	578
Hungary	935	2,287	46,687	1,318
Sri Lanka	890	2,196	47,092	844
Total	2,565	6,288	132,542	2,740

Figure 8: Measurements of data gathered from crawls of popular sites in selected under-served regions. Given numbers are counts of unique image and frame URLs.

(3) The region has seen a vast increase of Internet usage in the past decade[2].
(4) Had a popular sites listing on Alexa Top Sites.
(5) There existed at least one filter list for the region.
(6) We could purchase or gain access to a VPN with an exit point in the country.

Figure 6 presents the regions we selected for this evaluation, along with measurements of the existing best-maintained regional filter list. For each region we identified the best maintained filter list for the region by consulting both the EasyList selection of regional filter lists [18], and the filter lists indexed on a popular, crowd-sourced site of regional filter lists [8]. To further illustrate the lack of updates, Figure 7 illustrates how much activity there is on average each month in the respective GitHub repos between the selected regions and some popular filter lists.

4.2 Crawl Data Set

We next built a data set of popular websites and pages for each of the three selected regions. We use this data set for two purposes: first to approximate how internet users in these regions experience the web, and second to determine how much advertising content was being missed by existing content blocking options.

For each region, we first fetched the 1,000 most popular domains for the region, as determined by the Alexa Top Lists. Next, we purchased VPN access from ExpressVPN [6], a commercial VPN service, that provided an IP address in each region. Third, we configured a crawler to visit each domain and select two random child links with the same `eTLD+1`. We then configured our crawler to use our PageGraph-instrumented browser to visit the domain of each site and each selected child page, each for 30 seconds. All crawling was conducted from the VPN end point, to as closely as possible approximate how the page would run for a local visitor.

After 30 seconds, we recorded the PageGraph data for each page (note, the PageGraph data includes information about all network requests issued during the page's execution, in addition to the cause of each request). We also record all images and scripts fetched in the top-and-local frames (i.e. `<iframe>`s with the same domain as the top level frame) during each page's execution, along with screenshots of each remote child-frame (i.e. `<iframe>`s of third-party domains).

Figure 8 presents the results of our automated crawl. For each of the regions we encountered a significant number of non-responsive domains, which comprise the difference between the number in the "# Domains" column and the 1,000 domains identified by Alexa. While the 10% non-responsive rate for Hungary and Sri Lanka is inline with prior web-studies [31] that find around 11% error rate for automated crawls of the web, the even higher rate of non-responsive sites in Albania was surprising. On manual evaluation of a sample of these domains, we found a small number of cases were due to anti-crawler countermeasures or apparent IP blacklisting of the VPN end point. In a surprising number of cases though, domains seemed to be abandoned and hosting no web content at all. We note this as a point for future study.

Country	# Ad Images	%	# Ad Frames	%
Albania	2,553	6.6%	164	28.4%
Hungary	1,479	3.2%	209	15.9%
Sri Lanka	1,451	3.1%	296	35.1%
Total	5,483	4.1%	669	24.4%

Figure 9: Measurements of how many unique image and iframe ads are currently identified by existing filter lists (e.g. EasyList, EasyPrivacy, and the best maintained filter list for each region.).

Country	# Ad Images	%	# Ad Frames	%
Albania	440	17.2%	20	12.2%
Hungary	547	37%	10	4.8%
Sri Lanka	510	35.1%	17	5.7%
Total	1,497	27.3%	47	7%

Figure 10: Measurements of how many unique image and iframe ads the classifier described in Section 3.2 identified that *were not identified* as ads by existing filter lists.

4.3 Ads Identified by Existing Filter Lists

Next, we measured how successful existing filter lists are at blocking advertisements on popular sites in our selected regions. We treated this measurement as our baseline when measuring how much additional blocking benefit our approach provides. Figure 9 shows the number of ads identified by existing filter lists.

We measured the amount of ad resources identified by existing lists in two steps. First, we combined the best maintained regional filter list for each region (listed in Figure 6) with the two most popular "global" filter lists, EasyList and EasyPrivacy. Then, we applied these combined filter lists to the image and iframe requests encountered when crawling each region, using a popular AdBlock filter list library [23]. We then noted which images and iframe requests would be blocked by the current best filter lists available to people in each region.

4.4 Ads Identified by Hybrid Classifier

Next, we identified how many advertising images and iframes we missed by existing filter lists. We found a significant number of both; our classifier identified 1,497 ad images and 47 ad frames that were missed by existing filter lists. Put differently, our approach identified 27.3% more ad images, and 7% more ad frames, than existing filter lists.

We note that these figures only include ad images and frames missed by filter lists; we observed a significant amount of overlap

[2] Statistics gathered from `www.internetlivestats.com`

Country	Current Lists	Classifier	∪ Chains	Δ
Albania	2,850	460	511	18%
Hungary	1,819	557	586	32.2%
Sri Lanka	1,872	527	674	36%
Total	6,541	1,544	1,771	27.1%

Figure 11: Additions to filter lists when applying all steps of our methodology. "Current lists" gives the number of ad-resources found by existing filter lists, "classifier" describes ad-resources found by our hybrid classifier but not existing filter lists. "∪ chains" gives the number of new ad-resources found by applying our "up-stream" approach to ad-resources found by either current filter lists or the hybrid classifier. The "Δ" column gives the overall increase in identified ad-resources provided by our techniques, compared to existing filter lists.

between the two approaches. Figure 10 summarizes the additional ad resources our classifier identified.

We measured how many advertising images and frames current approaches miss in two steps. First, we identified each image or frame in the corresponding PageGraph graph data, and used that contextual information to extract the contextual features described in Section 3.2.2. Second, we used the pixel data of each resource to feed the perceptual part of the classifier. For images, we used the image file directly; for frames we used a screenshot of the frame taken during the crawling step.

4.5 Generalizing Rules with Request Chains

Next, we identified additional resources that should be blocked by examining the request chain for each ad image or frame, and finding the earliest point in each chain that could be blocked without breaking the page. We applied this "upstream request chain" blocking technique to both ad resources labeled by existing filter lists and ad resources newly identified by our hybrid classifier. Doing so allowed us to not only identify specific images and frames that should be blocked, but to programmatically identify the "upstream" libraries that caused those images and frames to be included.

We were able to identify 1,771 additional advertising URLs by analyzing the request chains in this manner, an improvement of 27.1% in advertisement blocking in these regions. We note that by following the methodology described in Section 3.5, targeting these additional resources will result in more generalizable filter rules by identifying both the individual ad image URLs *and* the ad libraries that determine what ads to load. The approach described in Section 3.5 also gives us a high degree of confidence that this "upstream" blocking will not break pages.

Figure 11 shows the final results of our regional filter list methodology when applied to the Albanian, Hungarian and Sri Lankan web regions. The "current lists" column presents the number of ad resources existing filter lists identify in each region in our data set. The "classifier" column gives the number of images and frames our hybrid classifier identifies as ad-related *that are not identified* by existing filter lists. The "∪ chains" column gives the total number of ad resources identified by applying the request chain approach (Section 3.4) to images and frames identified as advertisements by *either* existing filter lists or the hybrid classifier. The final "Δ" column gives the percent-increase in resources identified by our combined methodology, when compared to using only existing filter lists.

Country	Network rules
Albania	387
Hungary	551
Sri Lanka	372
Total	1310

Figure 12: Number of new filter list rules.

4.6 Generated Filter-Lists

Finally, we generated filter lists in AdBlock Plus format to block the advertising resources identified in the previous steps. We used the rule generation methodology described in Section 3.6 to generate 1310 new filter rules. We have made the filter lists available [10]. Counts of the total number of new rules for each region are presented in Figure 12.

5 DISCUSSION

In this section, we discuss broader issues related to the problem of filter list generation and ad blocking, including possible next steps and extensions for the described approach, and some limitations and concerns for future researchers to consider.

5.1 Ad Ambiguity

A reoccurring issue in identifying and blocking online advertisements is that many images and ads are context specific. An ad in one context might be core content in another. For example, an image of shoes with the name of the shoe maker might be perceived as an advertisement when positioned next to a news article, but the same image might be desirable when placed in the middle of a page on a shoes selling website.

We encountered an even more difficult case when labeling and debugging the pipeline described in this work. We found a movie sharing forum that used a number of banner ads (like the one presented in Figure 13) from elsewhere on the web as a table of contents, to show which movies had most recently been added on the site. In such cases, the "ad-ness" of an image is not ambiguous, its explicitly both an ad and desirable page content!

Crowd-sourced filter list generation approaches rely on the subjective intuition of list contributors to resolve such difficult situations. Programmatic solutions have no such option, and so must address a two tiered problem: first, how to identify images that look like advertisements, and second, how to model subjective user expectations of when an advertisement is desirable to users.

We find the problem presented by the intersection of these two issues is unaddressed by existing literature (current work included). Our resolution to this issue was "ads are ads"', and it is the job of ad blocking tools to block ads, and if a user is in a scenario where they wish to see and ad, they should disable the ad-blocking tool. How satisfying such an approach is will likely be task specific. Thankfully, the number of ambiguous ads we encountered was low enough that it did not affect the main focus of the work, but we mention it as an interesting area for future research.

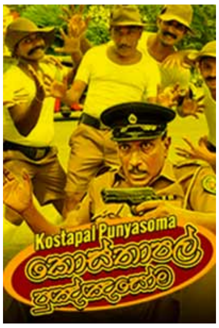

Figure 13: Example of an ambiguous ad image we encountered on a movie sharing forum in Sri Lanka as part of the its table of contents.

5.2 Alternative Image-Identification Oracles

This work used a unique hybrid approach for determining whether an image or a frame was an advertisement. This is only one of an infinite number of possible oracles the same pipeline could adopt. While we designed our approach to be conservative in identifying images (as described in Section 3.2), one could instead use a much more aggressive oracle, if one was willing to accept a greater false-positive rate in ad-identification, or was willing to accept sites breaking, for additional data savings and privacy protections.

In this sense, our oracle represents just one web use preference (less advertisements, but with a low tolerance for error). The same broad approach, as described in this paper, could be used with other oracles, such as those targeting just certain types of advertisements (i.e. blocking adult ads), or certain types of web content in general (i.e. blocking violent images).

While our goal in this work was to improve web browsing for people in under-served regions, the described approach is not specific to advertising. The identify-and-prune-the-request-chain approach could be helpful in addressing many web problems where human labelers are lacking.

5.3 Possible Extensions

We considered many additional features and approaches when designing the methodology in this work. Here, we briefly describe a variety of improvements we considered, but did not implement because of time, cost or complexity. We list them as possible suggestions to other researchers addressing similar problems.

Predicting Page Breakage. An important part of this work was generating and testing useful heuristics for whether blocking a script would break a page. The heuristics discussed in Section 3.5 have proven useful for us, but could be improved. One could, for example, also consider the number and type of Web API calls a script makes, whether the script sets or reads storage, or any number of other behavioral characteristics when trying to predict whether blocking a script would break a page.

Tracking Protections. This work improves blocking *advertisements* in under-served regions, but similar approach could be taken to target *tracking scripts.* Instead of building a classifier to determine if an image is an advertisement, one would instead need an oracle to determine whether a script was privacy violating. This might be an easier task, since determining the privacy implications of a script's execution is in many cases easier that predicting the subjective evaluation of whether an image is an advertisement.

Improving Other Filter Lists. The approach in this paper was designed to help web users in under-served regions. However, the same approach could likely be used to improve filter lists in general, including "global" popular ones like EasyList and EasyPrivacy. Though the marginal improvement would likely be lower, since the relative popularity of such sites likely means a higher percentage of ad resources have been identified by filter list contributors, our approach could still be useful in improving blocking on less popular, or frequently changing sites.

5.4 Limitations

Finally, we note some limitations of this approach, in the hopes that future work might address them. First, while our approach was successful on the three selected regions, its difficult to know if these findings would generalize to all under-served regions on the web. While such a measurement is beyond our ability to carry out, it would be interesting to better understand how similar web advertising is across the web generally.

Second, our approach relies on automated, manual crawls of websites to identify ad-related resources. It is possible that the kinds of advertisements reachable by automated tools are different from the kinds of advertisements humans experience when on parts of the web not reachable by crawlers, such as within web applications, behind paywalls, or within account-requiring portions of websites. This limitation is a subset of a larger open problem in web measurement, of understanding how well automated crawls approximate human user experiences.

Finally, the types of advertisements targeted in this work (i.e. image based web advertisements) are just one of many types of advertisements web users face. A partial list includes audio ads, video interstitials, native text ads, and interactive advertisements. If image- and frame- targeting ad blockers continue to become more popular, we can expect advertisers to adopt to these alternative advertising approaches. Researchers will in turn need to come up with new ad blocking techniques to preserve a usable, performant, privacy respecting web.

6 RELATED WORK

Below we cover the existing work related to filter lists (Section 6.1), resource blocking (Section 6.2), and the importance of different vantage points for web measurements (Section 6.3).

6.1 Filter Lists

In [41], Vastel et al. explored the accumulation of dead rules by studying EasyList, the most popular filter list. Results of their study show that the list has grown from several hundred rules, to well over 60,000 rules, within 9 years, when 90.16% of the resource blocking rules are not useful. Finally, authors, propose optimizations for popular adblocking tools, that allow EasyList to be applied on performance constrained mobile devices, and improve desktop performance by 62.5%.

Gugelmann et al. in [29] investigated how to detect privacy-intrusive trackers and services from passive measurements and propose an automated approach that relies on a set of web traffic features to identify such services and thus help developers maintaining filter lists. Pujol et al. in [38] used Adblock Plus filter lists for passive network classification. By analyzing data from a major European ISP authors show that 22% of the active users have Adblock Plus deployed. Also they found that 56% and 35% of the ad-related requests are blacklisted by EasyList and EasyPrivacy, respectively.

Iqbal et al., in [33], studied the anti-adblock filter lists that ad blockers use to remove anti-adblock scripts. By analyzing the evolution of two popular anti-adblock filter lists, authors show that their coverage considerably improved the last 3 years and they are able to detect anti-adblockers on about 9% of Alexa top-5K websites. Finally authors proposed a machine learning based method to automatically detect anti-adblocking scripts.

6.2 Resource Blocking

Iqbal et al., in [34], proposed AdGraph: a graph-based machine learning approach for detecting advertising and tracking resources on the web. Contrary to filter list based approaches AdGraph builds a graph representation of the HTML structure, network requests, and JavaScript behavior of a webpage, and uses this unique representation to train a classifier for identifying advertising and tracking resources. AdGraph can replicate the labels of human-generated filter lists with 95.33% accuracy.

In [39], Storey et al. discussed the future of ad blocking by modelling it as a state space with four states and six state transitions, which correspond to techniques that can be deployed by either publishers or ad blockers. They also proposed several new ad blocking techniques, including ones that borrow ideas from rootkits to prevent detection by anti-ad blocking scripts. Zhu et al., in [42], proposed ShadowBlock: a new Chromium-based ad-blocking browser that can hide traces of ad-blocking activities from anti-ad blockers. ShadowBlock leverages existing filter lists and hides all ad elements stealthily so anti-ad blocking scripts cannot detect any tampering of the ads (e.g. , absence of ad elements). Performance evaluation on Alexa top-1K websites shows that their approach successfully blocks 98.3% of all visible ads while only causing minor breakage on less than 0.6% of the websites.

Garimella et al. in [28] measured the performance and privacy aspects of popular ad-blocking tools. Their findings show that (i) uBlock has the best performance, in terms of ad and third party tracker filtering, and least privacy tracking. They also found that the time to load pages is not necessarily faster when using adblockers, and this happens due to additional functionality introduced by the adblocking tools. In [40], Din at al. proposed Percival: a deep learning based perceptual ad blocker that aims to replace filter list based adblocking. Percival runs within the browser's image rendering pipeline, intercepts images during page execution and by performing image classification, it blocks ads. Percival can replicate EasyList rules with an accuracy of 96.76% when it imposes a rendering performance overhead of 4.55%.

6.3 Internet Vantage Points

Selecting different vantage points to browse Internet from is a quite common technique in order to understand the different view of the web different users may have. Jueckstock et al. in [35] design and deploy a synchronized multi-vantage point web measurement study to explore the comparability of web measurements across different Internet vantage points. In [27] Fruchter et al. proposed a method for investigating tracking behavior by analyzing cookies and HTTP requests from browsing sessions from different countries. Results show that websites track users differently, and to varying degrees, based on the regulations of the country the visitor's IP is based in.

Iordanou et al. in [32] proposed a system for measuring how e-commerce websites discriminate between users. Authors consider several different motivations for discrimination, including geography, prior browsing behavior (e.g., tracking-derived PPI) of the user, and site A/B testing. They found that the first and third motivations explain more website "discrimination" than the second.

7 CONCLUSIONS

In this work we address the problem of augmenting filter lists for users of under-served, linguistically-small parts of the web. The approach described in this work is amenable to full automation, and with sufficient computation resources could be applied to any number of additional under-served populations of web users. Further, we expect the same approach could be used to block tracking-related resources too, improving privacy for under-served web users too.

The problem of poorly maintained filter lists in under-served regions is significant. First, the current predominant approach to filter list generation (i.e. crowd-sourcing) is poorly suited for these web-regions, which by definition have less users, and so smaller "crowds." Second, in many cases, under-served areas of the web target users with less income, and with less access to cheap, high speed data; the users who would benefit most from ad blocking are often the poorest served by current filter list generating strategies. Third, existing filter list generation strategies that *do not* rely on crowd-sourcing fail to consider web compatibility (i.e. breaking sites), leaving under-served users with the unappealing trade-off between data-draining, privacy-harming browsing, or, alternatively, breaking web sites.

This work proposes a novel approach for generating filter rules for under-served regions of the web. Our approach determines whether images and frames are advertisements by considering perceptual and contextual aspects of the underlying image (or frame), and then using deep browser instrumentation to determine where in the request chain we can optimally begin blocking requests.

We apply this approach to popular websites in three regions currently poorly served by crowd-sourced filter lists, Sri Lanka, Hungary, and Albania. Our approach is successful at improving blocking without breaking websites. We generate 1310 new filter list rules that identify 27.1% new advertising resources that should be blocked, improving blocking by 30.1% over the existing best options for these regions. We are also releasing our generated filter lists so that web users in these regions can benefit from them [10], along with the source code for our hybrid image classifier [7] and our PageGraph browser instrumentation [19]. We hope this work advances the goal of improving the web for all users, no matter their location or linguistic-community.

ACKNOWLEDGMENTS

This work was partly funded by the Swedish Foundation for Strategic Research (SSF) and the Swedish Research Council (VR).

REFERENCES

[1] [n.d.]. Adblock List for Albania. https://github.com/anxh3l0/blocklist. accessed: Oct-2019.

[2] [n.d.]. Adblock Plus filters explained. https://adblockplus.org/filter-cheatsheet/. accessed: Jan-2020.

[3] [n.d.]. Adblock Sri Lanka. https://github.com/miyurusankalpa/adblock-list-sri-lanka. accessed: Oct-2019.

[4] [n.d.]. Brazilian filterlist. https://raw.githubusercontent.com/easylistbrasil/easylistbrasil/filtro/easylistbrasil.txt. accessed: Oct-2019.

[5] [n.d.]. Chinese filterlist. https://easylist-downloads.adblockplus.org/easylistchina.txt. accessed: Oct-2019.

[6] [n.d.]. ExpressVPN. accessed: Oct-2019.

[7] [n.d.]. Filterlist Generator GitHub repo. https://github.com/brave-experiments/regional-filterlist-gen. accessed: Jan-2020.

[8] [n.d.]. FilterLists. https://filterlists.com/. accessed: Jan-2020.

[9] [n.d.]. French filterlist. https://easylist-downloads.adblockplus.org/liste$_{f}$r.txt. accessed: Oct-2019.

[10] [n.d.]. Generated Filter Lists. https://sites.google.com/site/longtailfilterlists/. accessed: Jan-2020.

[11] [n.d.]. German filterlist. https://easylist.to/easylistgermany/easylistgermany.txt. accessed: Oct-2019.

[12] [n.d.]. GitHub repo for German filter list. https://github.com/easylist/easylistgermany. accessed: Jan-2020.

[13] [n.d.]. GitHub repo for Hindi filter list. https://github.com/mediumkreation/IndianList. accessed: Jan-2020.

[14] [n.d.]. GitHub repo for Japanese filter list. https://github.com/k2jp/abp-japanese-filters. accessed: Jan-2020.

[15] [n.d.]. Hindi filterlist. https://easylist-downloads.adblockplus.org/indianlist.txt. accessed: Oct-2019.

[16] [n.d.]. hufilter. https://github.com/hufilter/hufilter/wiki. accessed: Oct-2019.

[17] [n.d.]. Japanese filterlist. https://raw.githubusercontent.com/k2jp/abp-japanese-filters/master/abpjf.txt. accessed: Oct-2019.

[18] [n.d.]. Other Supplementary Filter Lists and EasyList Variants. https://easylist.to/pages/other-supplementary-filter-lists-and-easylist-variants.html. accessed: Jan-2020.

[19] [n.d.]. PageGraph. https://github.com/brave/brave-core/wiki/PageGraph. accessed: Jan-2020.

[20] [n.d.]. Portugese filterlist. https://easylist-downloads.adblockplus.org/easylistportuguese.txt. accessed: Oct-2019.

[21] [n.d.]. Russian filterlist. https://easylist-downloads.adblockplus.org/advblock.txt. accessed: Oct-2019.

[22] [n.d.]. The GraphML File Format. http://graphml.graphdrawing.org/. accessed: Jan-2020.

[23] Ben Livshits Andrius Aucinas. 2019. Brave Improves Its Ad-Blocker Performance by 69x with New Engine Implementation in Rust. https://brave.com/improved-ad-blocker-performance/.

[24] Sruti Bhagavatula, Christopher Dunn, Chris Kanich, Minaxi Gupta, and Brian Ziebart. 2014. Leveraging machine learning to improve unwanted resource filtering. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*. ACM, 95–102.

[25] Business Insider Intelligence. 2017. 30% of all Internet users will ad block by 2018. https://www.businessinsider.com/30-of-all-internet-users-will-ad-block-by-2018-2017-3.

[26] V. Dudykevych and V. Nechypor. 2016. Detecting third-party user trackers with cookie files. In *2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S T)*. 78–80. https://doi.org/10.1109/INFOCOMMST.2016.7905341

[27] Nathaniel Fruchter, Hsin Miao, Scott Stevenson, and Rebecca Balebako. 2015. Variations in Tracking in Relation to Geographic Location. *CoRR* abs/1506.04103 (2015). arXiv:1506.04103 http://arxiv.org/abs/1506.04103

[28] Kiran Garimella, Orestis Kostakis, and Michael Mathioudakis. 2017. Ad-blocking: A Study on Performance, Privacy and Counter-measures. In *WebSci*. ACM, 259–262.

[29] David Gugelmann, Markus Happe, Bernhard Ager, and Vincent Lenders. 2015. An automated approach for complementing ad blockers' blacklists. *Proceedings on Privacy Enhancing Technologies* 2015, 2 (2015), 282–298.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[31] Luca Invernizzi, Kurt Thomas, Alexandros Kapravelos, Oxana Comanescu, Jean-Michel Picod, and Elie Bursztein. 2016. Cloak of visibility: Detecting when machines browse a different web. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 743–758.

[32] Costas Iordanou, Claudio Soriente, Michael Sirivianos, and Nikolaos Laoutaris. 2017. Who is Fiddling with Prices?: Building and Deploying a Watchdog Service for E-commerce. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. ACM, New York, NY, USA, 376–389. https://doi.org/10.1145/3098822.3098850

[33] Umar Iqbal, Zubair Shafiq, and Zhiyun Qian. 2017. The Ad Wars: Retrospective Measurement and Analysis of Anti-adblock Filter Lists. In *Proceedings of the 2017 Internet Measurement Conference* (London, United Kingdom) *(IMC '17)*. ACM, New York, NY, USA, 171–183. https://doi.org/10.1145/3131365.3131387

[34] Umar Iqbal, Zubair Shafiq, Peter Snyder, Shitong Zhu, Zhiyun Qian, and Benjamin Livshits. 2018. AdGraph: A Machine Learning Approach to Automatic and Effective Adblocking. *CoRR* abs/1805.09155 (2018). arXiv:1805.09155 http://arxiv.org/abs/1805.09155

[35] Jordan Jueckstock, Shaown Sarker, Peter Snyder, Panagiotis Papadopoulos, Matteo Varvello, Benjamin Livshits, and Alexandros Kapravelos. 2019. The Blind Men and the Internet: Multi-Vantage Point Web Measurements. *CoRR* abs/1905.08767 (2019). arXiv:1905.08767 http://arxiv.org/abs/1905.08767

[36] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. 2012. Knowing your enemy: understanding and detecting malicious web advertising. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. 674–686.

[37] Arvind Parmar, Malcolm Toms, Costa Dedegikas, and Christopher Dickert. [n.d.]. Adblock Plus Efficacy Study. http://www.sfu.ca/content/dam/sfu/snfchs/pdfs/Adblock.Plus.Study.pdf. accessed: Oct-2019.

[38] Enric Pujol, Oliver Hohlfeld, and Anja Feldmann. 2015. Annoyed Users: Ads and Ad-Block Usage in the Wild. In *Proceedings of the 2015 Internet Measurement Conference* (Tokyo, Japan) *(IMC '15)*. ACM, New York, NY, USA, 93–106. https://doi.org/10.1145/2815675.2815705

[39] Grant Storey, Dillon Reisman, Jonathan Mayer, and Arvind Narayanan. 2017. The future of ad blocking: An analytical framework and new techniques. *arXiv preprint arXiv:1705.08568* (2017).

[40] Zain ul Abi Din, Panagiotis Tigas, Samuel T. King, and Benjamin Livshits. 2019. Percival: Making In-Browser Perceptual Ad Blocking Practical With Deep Learning. *CoRR* abs/1905.07444 (2019). arXiv:1905.07444 http://arxiv.org/abs/1905.07444

[41] Antoine Vastel, Peter Snyder, and Benjamin Livshits. 2018. Who Filters the Filters: Understanding the Growth, Usefulness and Efficiency of Crowdsourced Ad Blocking. *CoRR* abs/1810.09160 (2018). arXiv:1810.09160 http://arxiv.org/abs/1810.09160

[42] Shitong Zhu, Umar Iqbal, Zhongjie Wang, Zhiyun Qian, Zubair Shafiq, and Weiteng Chen. 2019. ShadowBlock: A Lightweight and Stealthy Adblocking Browser. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. ACM, New York, NY, USA, 2483–2493. https://doi.org/10.1145/3308558.3313558